(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 1 365 538 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
26.11.2003 Bulletin 2003/48

(51) Int Cl.[7]: **H04L 12/18**, H04L 29/06

(21) Application number: **03253107.1**

(22) Date of filing: **19.05.2003**

(84) Designated Contracting States:
AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HU IE IT LI LU MC NL PT RO SE SI SK TR
Designated Extension States:
AL LT LV MK

(30) Priority: **23.05.2002 EP 02011310**

(71) Applicant: **Marchand, Benoit**
Montreal, Quebec H3R 3E1 (CA)

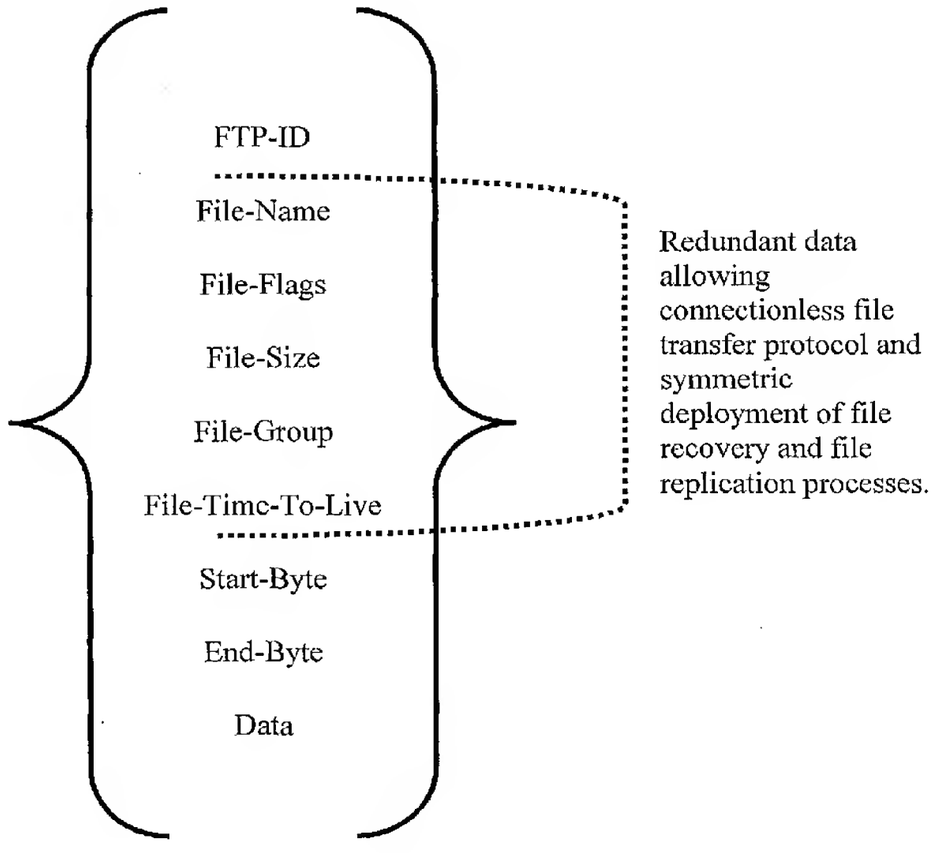
(72) Inventor: **Marchand, Benoit**
Montreal, Quebec H3R 3E1 (CA)

(74) Representative:
McLeish, Nicholas Alistair Maxwell et al
Boult Wade Tennant
Verulam Gardens
70 Gray's Inn Road
London WC1X 8BT (GB)

(54) Implementing a scalable, dynamic, fault-tolerant, multicast based file transfer and asynchronous file replication protocol

(57) Apparatus and method to improve the speed, scalability, robustness and dynamism of multicast data transfers to remote computers. Many Grid Computing applications, such as Genomics, Proteomics, Seismic, Risk Management, etc, require a priori transfer of sets of files or other data to remote computers prior to processing taking place. Existing multicast and data transfer protocols are static and can not guarantee that all nodes will contain a copy of the replicated data or files. The fully distributed data transfer and data replication protocol of the invention permits transfers which minimize processing requirements on master transfer nodes by spreading work across the network. The result is higher scalability than current centralized protocols, more dynamism and allows fault-tolerance by distribution of functionality. The ability to distribute the protocol is simplified through our innovative symmetric-connectionless data transfer protocol.

Figure 1: Symmetric-Connectionless File Transfer Protocol Primitive



EP 1 365 538 A1

Printed by Jouve, 75001 PARIS (FR)

Description

FIELD OF THE INVENTION:

[0001] The present invention relates to transferring and replicating data among geographically separated computing devices, and, in particular, to implementing a multicast file transfer protocol to transfer files more rapidly, robustly and to more computing devices than current methods permit. In addition, the invention can be used to asynchronously maintain a set of replicated files throughout computer failures and introduction of new computers into the network.

BACKGROUND OF THE INVENTION:

[0002] Grid Computers, Computer Farms and similar Computer Clusters are being used to deploy a novel type of parallel applications based on concurrent independent tasks related only by their individual contribution to a global problem's resolution. Until now all parallel applications were based on splitting a single task into a multitude of collaborating subtasks (ie OpenMP, PVM, MPI, etc). However, in some application areas users have recently started to split single large problems into a multitude of sub problems which can be resolved independently of one another. This methodology allows higher scalability and permits the use of Grid Computing techniques and the use of cost efficient computing solutions (ie clusters), but requires that the necessary data files first be replicated to the remote nodes prior to the computation taking place. It is this problem of replicated data transfers that our invention addresses.

[0003] Existing art to address data file transfer falls into three categories.

[0004] First, tasks can make use of on-demand file transfer apparatus, better known as file servers. For problems where file access is minimal, this type of solution works as long as the cluster size (ie number of remote computers) is limited to a few hundred. For large and frequent file accesses, this solution does not scale beyond a handful of nodes. Moreover, if entire data files are accessed by all nodes, the total amount of data transfer will be N times that of a single file transfer (where N is the number of nodes). This waste of network bandwidth limits scalability and penalizes computational performance as the nodes are blocked waiting for remote data.

[0005] Second, users or tasks can manually transfer files prior to execution though a point-to-point file transfer protocol. There are three types of point-to-point protocols. Standard file transfer protocols (ie ftp, tftp) where one file is transferred to one remote node, one packet at a time. Sliding window file transfer protocols, such as the "parallel file transfer protocol" from Donald J. Fabozzi II where multiple packets transit concurrently on their way to a single remote node. And parallel file transfer protocols (ex HPSS PFTP) where multiple point-to-point file transfers operate concurrently. While these methods improve network bandwidth utilization over demand based schemes, the final result is the same: a file is transferred "N" times over the network when replicating information unto "N" remote computers. Moreover, additional file transfers must continually be initiated to cope with the constantly varying nature of large computer networks (ie new nodes being added to increase a cluster or Grid size or to replace failed or obsolete nodes).

Third, users or tasks can manually transfer file prior to execution through a multicast (or broadcast) file transfer protocol (ex StarBurst SMFTP). In this scheme each file fragment sent over the network is simultaneously read by all participating remote computers. Hence network bandwidth usage is limited to the same amount of data traffic as for a single point-to-point file transfer. This is currently the most frequent scheme used to resolve problems having been split into multiple concurrent independent tasks as described above. However, this form of apparatus is imperfect. For instance, error recovery is concurrent to the multicast phase. This imposes an increased workload on the master file server node and eventually will limit scalablity. These schemes also are based on the notion of node registration, where prior to the multicast phase, all active and participating remote computers must register to participate in a transfer request. Hence, new nodes being booted during or after the multicast transfer phase will not be participating in the effort to replicate files. Another drawback is that registered computers which crash during the multicast phase can not join back the transfer group after reboot. Finally, these schemes can not survive through a crash on the master file server (ie the computer which performs the multicast file transfer). These sum of these limitations is that current multicast file transfer art work fail at their task of insuring correct file replication among all participating remote computers in a normal setup of dynamic and error prone network of computers. They lack the fault-tolerance, ability to handle dynamic registration, scalability to tens of thousands of nodes and capability to persist with the file replication effort once the master transfer process terminates.

SUMMARY OF THE INVENTION:

[0006] The object of the present invention is to implement a multicast data transfer apparatus which keeps operating through computer failures, allows data replication scalability to very large size networks, and, which persists transferring data to newly introduced nodes even after the master data transfer process has terminated.

[0007] The terms "computer" and "node" used in the description of the present invention must be understood in the broadest sense, as they can include any computing device or any electronic appliance including a computing device, such as for example a personal computer,

a cellular phone, a PDA, etc., which is or can be connected to any one type of network.

[0008] The term data transfer used in the description of the present invention must be understood in the broadest sense, as it can include full and partial data transfers. That is, it relates to transfers where an entire data entity (e.g. file) is transferred at once, as well as situations where selected segments of a data entity are transferred at some point. An example of the latter case is a data entity being transferred in its entirety and at a later time, selected segments of the data entity are being updated.

[0009] Briefly stated, the present invention ensures the correct replication of sets of files or any other data, for instance in a network of computers, in spite of network failures, computer crashes, or the introduction of new computers in the network.

[0010] The present invention innovates in the following areas:

1. symmetric-connectionless data transfer protocol allowing stateless data transfers (ie no need for a centralized master data transfer engine to maintain individual state information about participating nodes);
2. separation of the multicast data transfer phase and the point-to-point error recovery phase performed by two independent protocol engines;
3. distributed data transfer protocol where all participating remote computers can collaborate in the error recovery and data replication phases;
4. use of the recovery phase protocol to enable crashed computers to complete data transfers upon reboot;
5. use of the recovery phase protocol to enable newly introduced nodes to perform asynchronously recent data transfers having occurred before they became operational (ie data replication);
6. automatic removal of replicated data once they reach their pre-set life span;
7. fault-tolerance of the master data transfer process;
8. dynamically adaptable peer process selection mechanism through a random number and modulus calculation scheme;
9. full and partial (ie segments of files) data transfers are supported through the same apparatus.

[0011] The apparatus and method according to the invention improve the speed, scalability, robustness and dynamism of multicast data transfers to remote computers. Many Grid Computing applications, such as Genomics, Proteomics, Seismic, Risk Management, etc, require a priori transfer of sets of files or other data to remote computers prior to processing taking place. Existing multicast and data transfer protocols are static and can not guarantee that all nodes will contain a copy of the replicated data or files. The fully distributed data transfer and data replication protocol of the invention permits transfers which minimize processing requirements on master transfer nodes by spreading work across the network. The result is higher scalability than current centralized protocols, more dynamism and allows fault-tolerance by distribution of functionality. The ability to distribute the protocol is simplified through our innovative symmetric-connectionless data transfer protocol.

[0012] In particular, the present invention is preferably embodied by a method to using a file transfer protocol to transfer, without regards to a user's privilege, files between remote computers, comprising:

1. segmenting a file into a number of data packets to be multicast (or broadcasted) over a network of computers;
2. recording in a log at each receiving computer the segments of the transferred file already received and those still missing;
3. rebuilding the transferred file by writing received data packets at their original respective location in the file using direct access IO;
4. transmitting by a multicast, or broadcast, apparatus said packets over a network of computers;
5. recovering of missing, incomplete or corrupted data packets by means of a distributed transfer recovery apparatus independent from the transfer apparatus used initially to multicast the data packets;
6. completing of interrupted file transfers by cause of node failure upon reboot by means the recovery apparatus;
7. pursuing file transfers in spite of root transfer node failure by the automatic selection of an alternate multicast root transfer node;
8. synchronizing replicated files upon reboot, or the addition in the network, of a node by means of the recovery apparatus;
9. removing partially transferred files on the remote nodes upon canceling or aborting the file transfer request by the user, an operator or a system crash of the requesting node;
10. determining the number of operational nodes which are in the process of completing an in-progress file transfer or have already completed a file transfer;
11. removing automatically replicated files which have exceeded their preset life-span;
12. selecting peer processes (transfer master selection, transfer error recovery and file replication) through a dynamically adaptable random number and modulus calculation scheme.

BRIEF DESCRIPTION OF THE DRAWINGS:

[0013]

Figure 1 illustrates the symmetric-connectionless

file transfer protocol primitive;
Figure 2 illustrates the layout of the broadcast/multicast file transfer process;
Figure 3 depicts the layout of the transfer error recovery and file replication process;
Figure 4 shows the user interface process protocol finite state machine;
Figure 5 shows the file transfer master process protocol finite state machine;
Figure 6 illustrates the file transfer slave process protocol finite state machine;
Figure 7 shows the multicast/broadcast master process protocol finite state machine;
Figure 8 depicts the forwarder slave process protocol finite state machine;
Figure 9 shows the transfer error recovery slave process protocol finite state machine;
Figure 10 illustrates the file replication slave process protocol finite state machine;
Figure 11 shows the distributed selection mechanism.

DETAILED DESCRIPTION OF THE INVENTION:

[0014] Figure 1 summarizes the protocol primitive used to implement the symmetric-connectionless file transfer protocol. This protocol primitive is said to be connectionless (ie redundant) because it contains all information required to perform a file transfer in every data packet exchange. Indeed file name, file flags, life span, file size, etc are duplicated in each data packet. This information redundancy consumes less than 5% of packet space (Ethernet MTU of 1500bytes), but allows remote computers to easily "jump in" to any file transfer multicast phase without prior registration phase. Moreover, it allows simple and efficient error recovery and file synchronization for newly introduced nodes and out-of-order processing of data packets. The data transfer primitive is further said to be symmetric because it can be used by the master file transfer process (during the multicast transfer phase) or by any other participating nodes (for error recovery or file replication purposes).

[0015] Figure 2 shows the different processes layout to complete a multicast file transfer. A user interface process is launched by a user or automated tool to reach all active file transfer master processes and initiate the multicast file transfer. The scope of interaction between these two process types is defined by the geographic coverage of the first multicast/broadcast group. One file transfer master process is selected to proceed to the actual multicast to all active file transfer slave processes reachable by in the second multicast/broadcast group.

[0016] Figure 3 depicts the types of peer-to-peer (ie symmetric) exchanges among file transfer slave processes during a file transfer error recovery phase or a file replication phase. The geographic scope is delimited by the second multicast/broadcast group coverage.

[0017] Figures 4 through 10 show the finite state machines used to implement the multicast/broadcast file transfer and file replication protocols for the user interface, file transfer master and file transfer slave processes and their related sub-processes. The mode of operation can allow multiple concurrent multicast/broadcast file transfers and overlapping of multicast/broadcast file transfer, transfer error recovery and file replication phases. Fault-tolerance, scalability and dynamism are achieved through real-time peer selection and communication persistence.

[0018] Referring to Figure 1, all preceding File Transfer Protocol art is based on the notion of client-server connections or registrations. This requirement prevents dynamic client participation in file transfer activities. It further enforces strict delivery packet ordering. Finally, it necessitates a complex reconnection mechanism. Our file transfer protocol is, by opposition, based on a connectionless model where, without any preceding protocol exchange, file fragments can be exchanged among cooperating processes. Hence at the receiving end processes can jump into any ongoing file transfer exchange at any moment in time, and count on the transfer error recovery protocol to retrieve earlier packets, or missing packets alike. Furthermore, by splitting multicast transfer and recovery transfer phases, connectionless data exchanges allow any cooperating process to participate in error recovery and file replication, thus the symmetric nature of our apparatus. Symmetry also inherently implies higher scalability, since any number of processes may contribute to the recovery phase (the bottleneck of preceding point-to-point recovery arts), and fault-tolerance. Finally, a symmetric protocol allows asynchronous activities, past the normal termination of the multicast file transfer phase. This feature allows the implementation of a file replication mechanism where newly added or rebooted nodes may contact cooperating processes to synchronize with past file transfer activities.

[0019] Figure 2 represents the interconnection of processes in our apparatus. There are three process level components: the user interface, the file transfer master and the file transfer slave processes.

[0020] The user interface is mandated with establishing, and maintaining established, a link with any one of the active file transfer masters and sending the file fragments. The link is established by multicasting (or broadcasting) a request on a predefined communication port (socket port number) and selecting one of the active file transfer master. The presence of multiple file transfer masters and our communication protocol allows fault-tolerance, that is, the multicast file transfer will continue through file transfer master processes failures as long as there is still at least one active file transfer master. Moreover redundant file transfer master allows for concurrent multicast file transfers. A serialization or token mechanism may be added to prevent network saturation by limiting the number of simultaneous file transfers.

[0021] Once a file transfer master is selected to per-

form the multicast file transfer, it forks a child process to take over the multicast (or broadcast) transfer phase, allowing a single file transfer master to handle multiple transfer requests simultaneously. The child process then forwards all file fragments over the network to predetermined communication port for the benefit of all participating file transfer slave processes. Active file transfer slaves pick up the file fragments from the network and write them at their appropriate location in the target replicated file.

[0022] Figure 3 shows the sort of activities, among file transfer slave processes, which may persist after the multicast transfer phase has terminated. For instance, cooperating file transfer slaves may assist each other in an error recovery phase, forwarding file fragments to other slaves having missed some file fragments or received corrupted file fragments. A simple extension of this error recovery protocol allows for newly introduced nodes, running a file transfer slave, to catch up on earlier file transfers and (re)build their set of locally replicated files.

[0023] The selection mechanism, Figure 11, used by a user interface process to elect a file transfer master or by a file transfer slave process to choose another file transfer slave process to perform file replication or error recovery is based on a novel random number and modulus calculation scheme. Prior distributed computing methods to perform election are based on NxN message exchanges. This NxN problem resolution creates network communication bottlenecks in large networks with many elections to process and physically prevents scaling to tens of thousands of nodes. Moreover it requires an a priori knowledge of the network topology and number of participants. In our scheme, a partner selection, among a large set of cooperating candidates, is performed by performing a multicast (or broadcast) of a random number and a modulus number. Upon reception, likely candidates calculate two new random numbers. The first random number is applied the received modulus number and if the result matches the received random number, the second generated random number is sent back. The election originator accumulates returned answers for a limited amount of time and selects the candidate with the smaller returning random number. This scheme is made adaptative by varying the modulus number in order to reduce or increase the number of respondents. The modulus number to use for a new election round is based on the number or respondents from past requests, and initially is set to "1" (forces everybody to respond).

[0024] Figure 4 depicts the user interface process protocol finite state machine. The initial step is to select a file transfer master process to send file fragments to. This phase fails if no file transfer master processes are reachable. The transfer of file fragments begins and proceeds until all fragments have been transferred. Should the selected file transfer master process stop responding, a new election round is initiated and transfer may proceed from where it was interrupted.

[0025] The file transfer master process protocol finite state machine, Figure 5, is quite minimal; it replies to selection requests and, once selected, spawns a child process to conduct the actual multicast (or broadcast) file transfer phase. The multicast (or broadcast) file transfer process protocol finite state machine (Figure 7) consists in forwarding all file fragments received from the user interface process to all participating file transfer slave processes. Should the user interface process stop responding, the multicast file transfer process notifies all file transfer slave processes and terminates. The protocol may be extended to perform a file transfer completion check with all remote file transfer slave processes.

[0026] The file transfer slave process protocol finite state machine shown in Figure 6 implements the multicast file transfer reception side, the transfer error recovery mechanism and further contains two optional protocol extensions for file transfer completion and file replication. Single message exchange requests, such as completion check, transfer abort request, file replication or error recovery selection requests and reception of file fragments are handled directly by the slave process. All other tasks, such as assisting another slave to recover file fragments, or initiating a recovery procedure or file replication upon boot are handled in individual sub-processes. Consequently, a single file transfer slave process can handle multiple simultaneous file transfers and file transfer recovery procedures or can assist concurrently more than one slave process to recover missing or corrupted file fragments. The optional protocol extensions are file completion check and file replication procedure.

[0027] The file transfer forwarding process, Figure 8, consists in forwarding requested file fragments to the originating file transfer slave process until no further requests are received during a preset period of time.

[0028] Figure 9 shows the file transfer recovery process protocol finite state machine. After an initial selection phase, to locate a cooperating file transfer slave process, requests to forward missing (or corrupted) file fragments are sent out to the selected slave process. Cooperating processes respond to a forwarding request only if they possess a proper copy of the file fragment requested. If no cooperating slave process can be selected (ie no other slave process contains the requested file fragment) the incomplete file is removed and the recovery terminates. Forwarded file fragments, once received, are written in their correct location in the target file.

[0029] The overall multicast file transfer and recovery mechanism described so far can be further extended to perform automatic file replication as depicted in Figure 10 (file replication process protocol finite state machine). Upon startup a file transfer slave process can spawn a sub-process to perform asynchronously the file replication procedure. File replication serves two purposes: complete upon boot interrupted file transfers and perform file transfers that have occurred while the file

transfer slave process was non operational. The protocol starts by initiating a selection procedure to locate a cooperating file transfer slave process. This cooperating process serves the purpose to determine which file transfers occurred while the requesting slave process was non operational. Afterwards each file transfer missed, or interrupted (these can be determined locally from the file fragments stored) is completed using the normal file recovery protocol engine (either in an independent sub-process or not, depending on the implementation).

[0030] The combination of persistent connectionless requests and distributed selection procedure allows for scalability and fault-tolerance since there is no need for global state knowledge to be maintained by a centralized entity (or replicated entities). Furthermore it allows to build a light weight protocol which can be implemented efficiently even on appliance type devices. The use of multicast (or broadcast) minimizes network utilization, allowing higher aggregate file transfer rates and enabling the use of lesser expensive networking equipment (which in turn allows the use of lesser expensive nodes). The separation of multicast file transfer and recovery file transfer phases allows the deployment of a distributed file recovery mechanism that further enhances scalability and fault-tolerance properties. Finally, the independent file transfer recovery mechanism can be used to implement an asynchronous file replication apparatus, where newly introduced nodes (or rebooted nodes) can perform file transfers which occurred while they were non operational and after the completion of the multicast file transfer phase.

[0031] In its preferred embodiment, the present invention is applied to file transfer and file replication. The one skilled in the art will however recognize that the present invention can be applied to the transfer, replication and/or streaming of any type of data.

Claims

1. Method to transferring data between computing devices comprising a data transfer phase using a multicast and/or broadcast transfer protocol and further comprising an error recovery phase for recovering corrupted or missing data, **characterized in that** corrupted or missing data can be recovered from any one of said computing devices.

2. Method according to the preceding claim, said transfer protocol being connectionless.

3. Method according to one of the preceding claims, wherein said recovery phase is performed independently from said transfer phase.

4. Method according to one of the preceding claims, said recovery phase being used for transferring already transferred data from one of said computing devices to a newly connected computing device.

5. Method according to one of the preceding claims, said recovery phase being used for completing interrupted data transfers.

6. Method according to one of the preceding claims, said data being segments of a file.

7. Method according to one of the preceding claims, further comprising recording the received data in a log at each computing device of said computing devices receiving data.

8. Computing device adapted to performing the method of one of the preceding claims.

9. Computer program product, directly loadable into the internal memory of a computing device, comprising software code portions for performing the method of one of the claims 1 to 7.

Figure 1: Symmetric-Connectionless File Transfer Protocol Primitive

Each FTP Data or User Data packet contains:

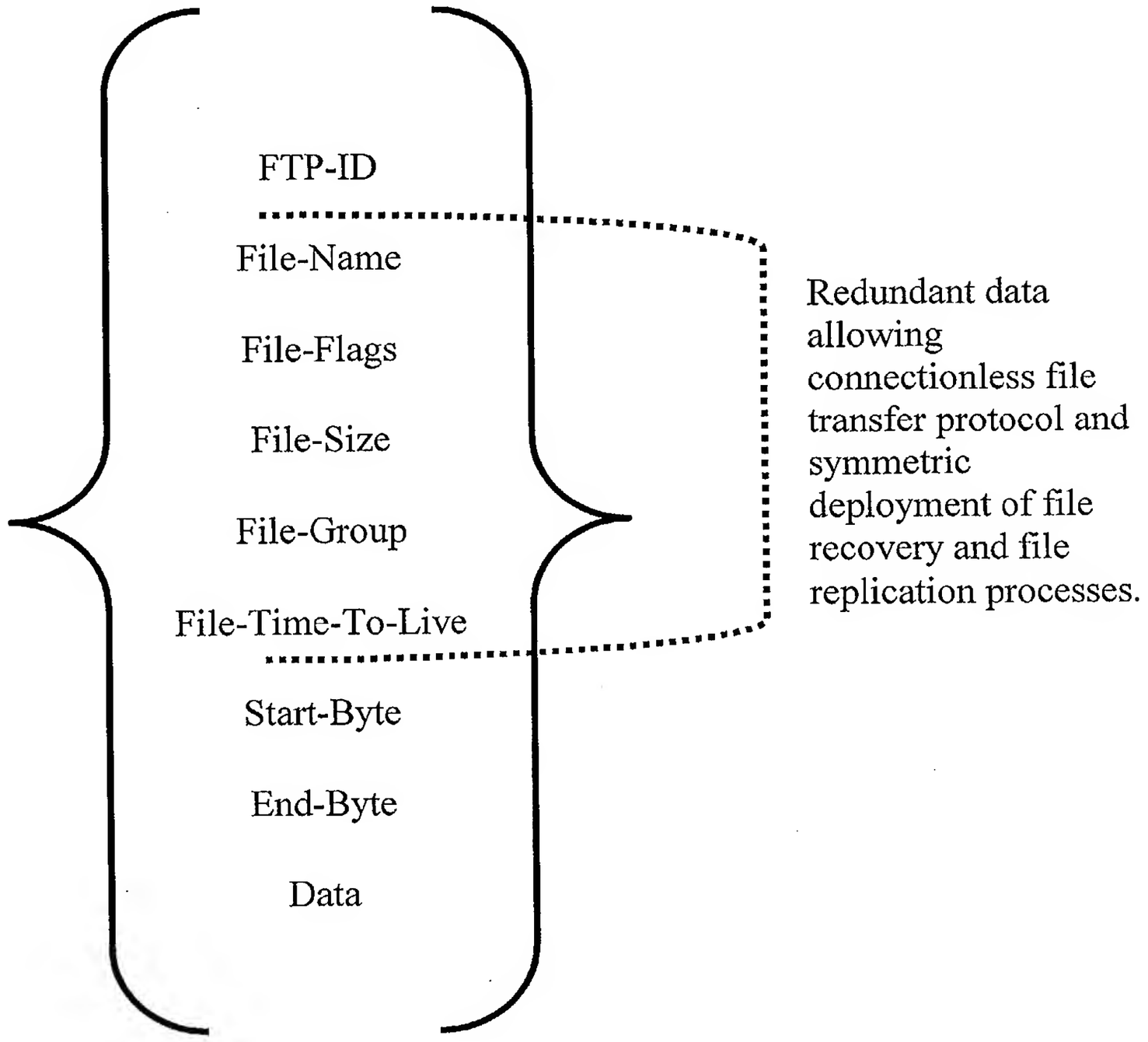


Figure 2: Broadcast/Multicast File Transfer Process Layout

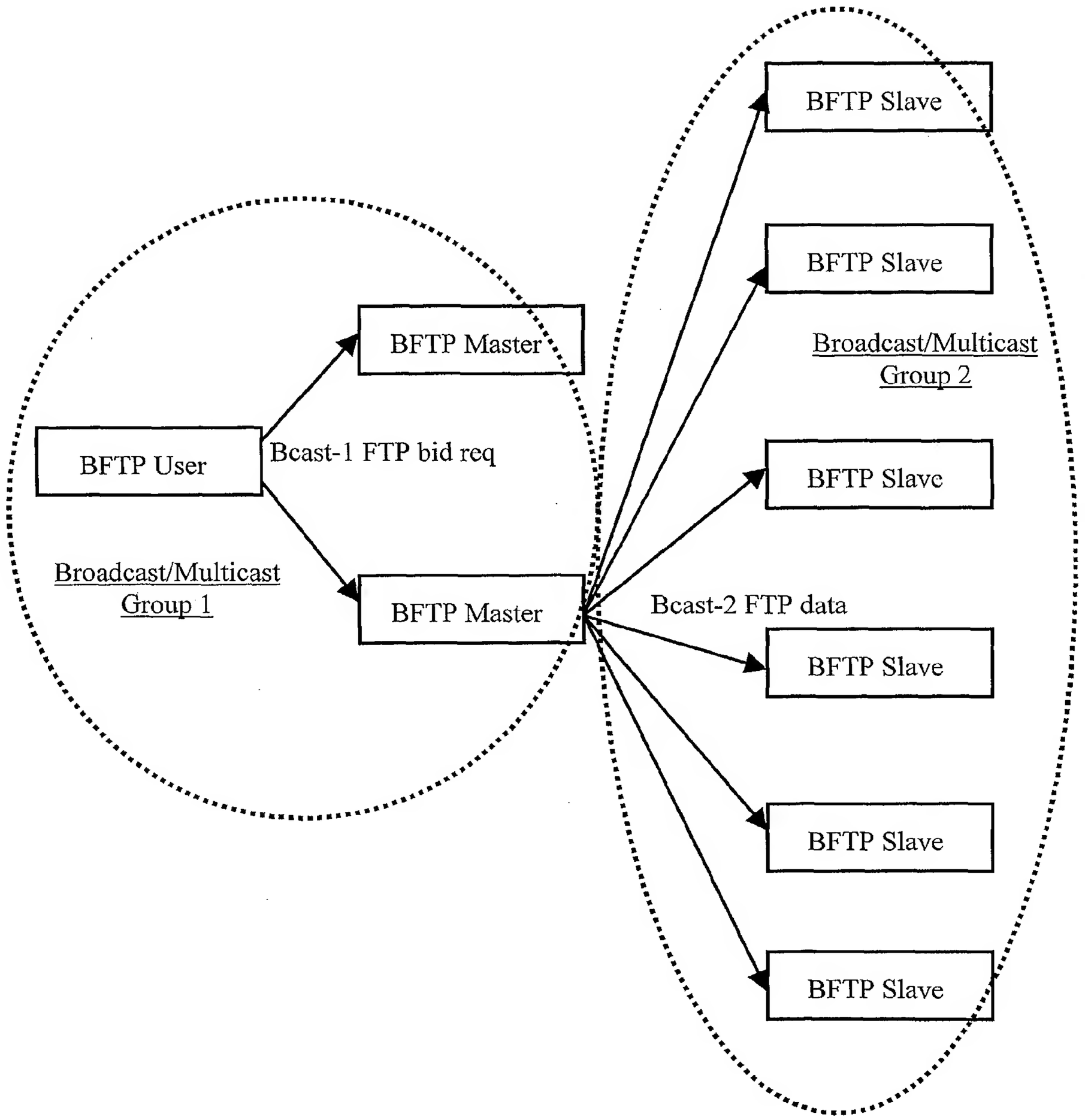


Figure 3: Transfer Error Recovery and File Replication Process Layout

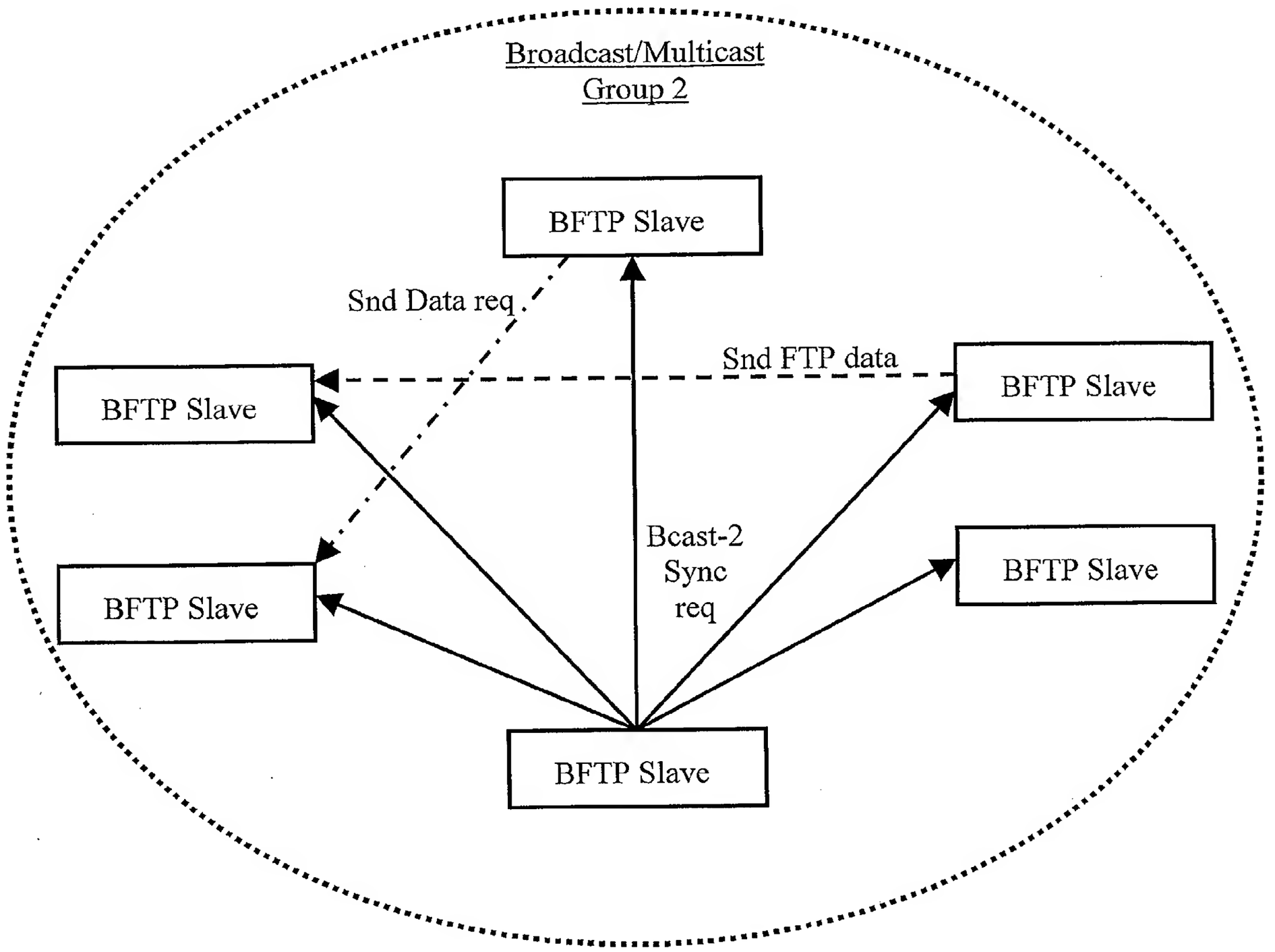


Figure 4: User Interface Process Protocol Finite State Machine

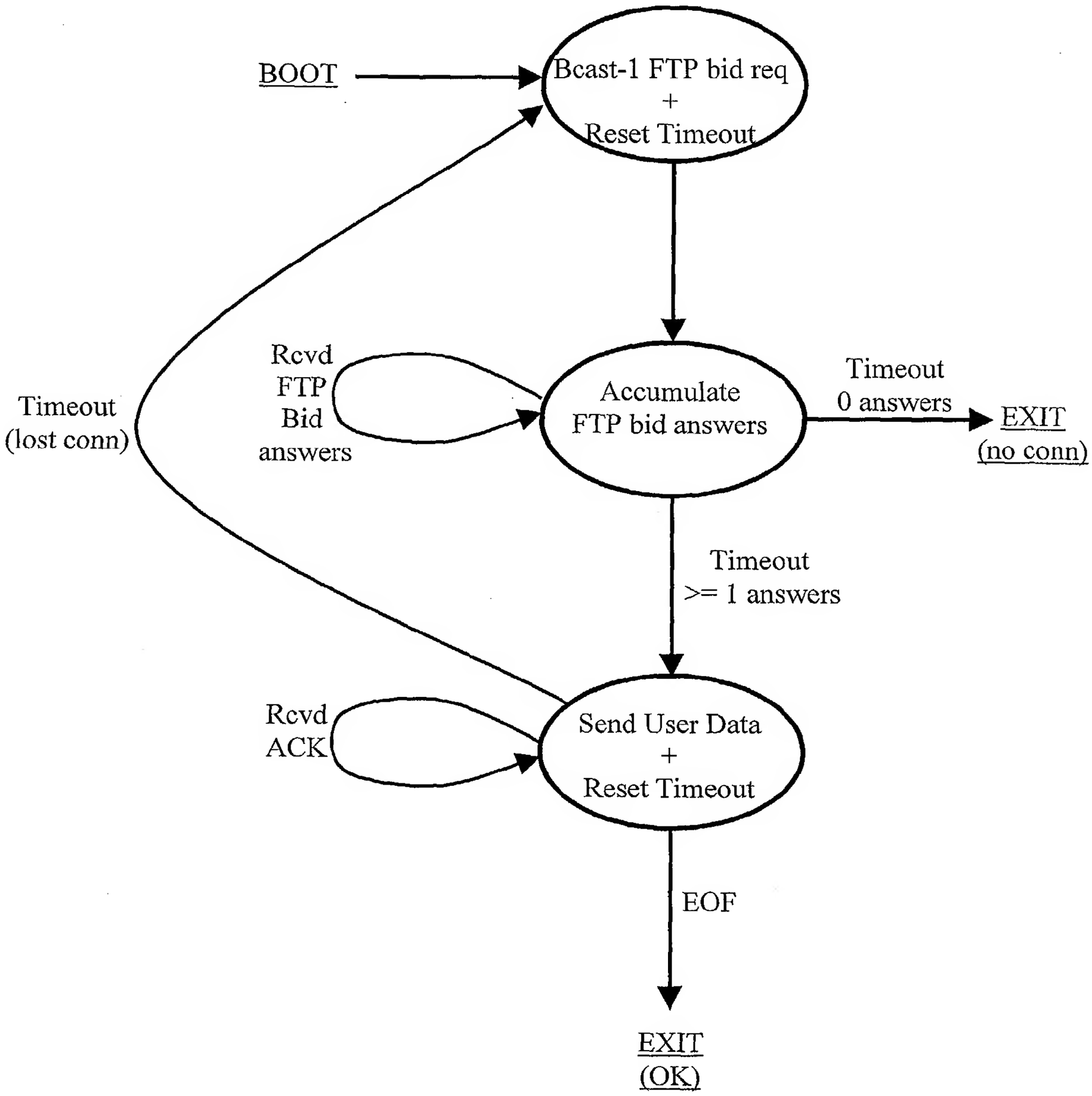


Figure 5: File Transfer Master Process Protocol Finite State Machine

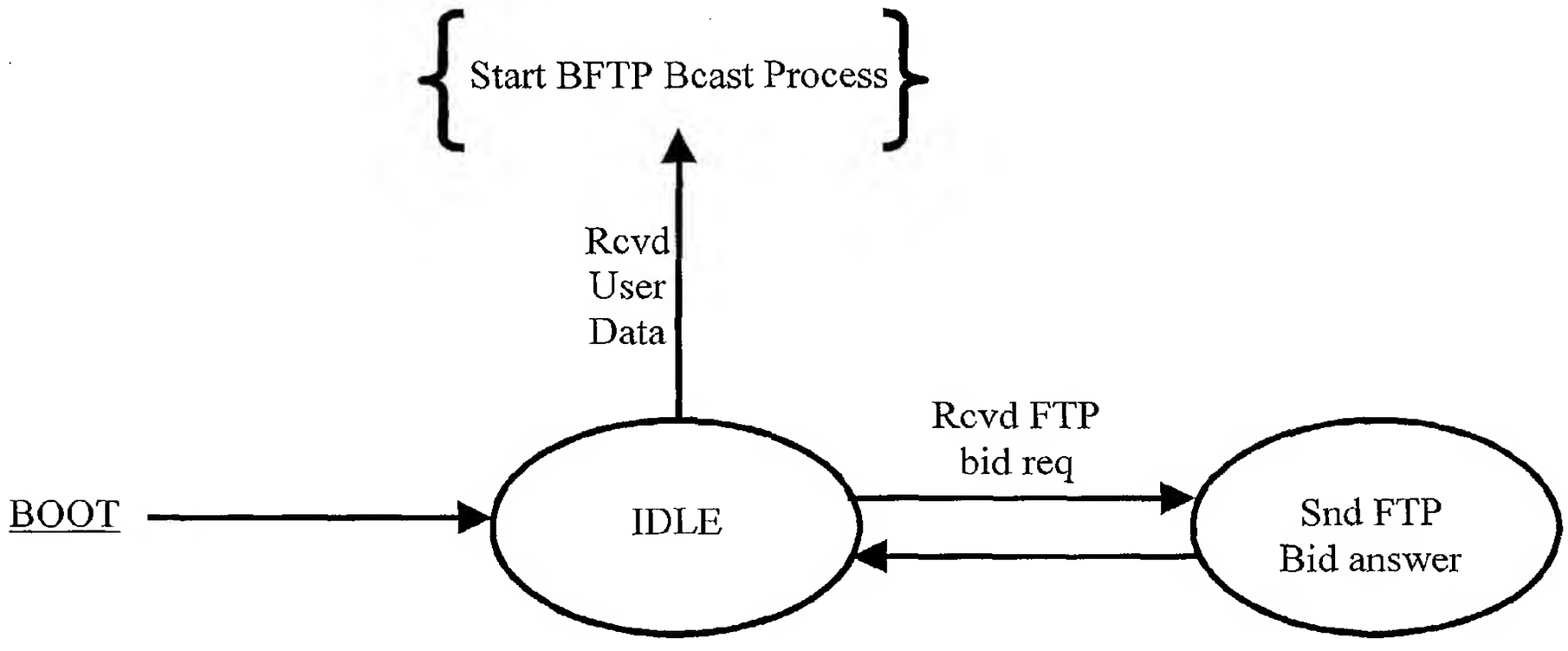


Figure 6: File Transfer Slave Process Protocol Finite State Machine

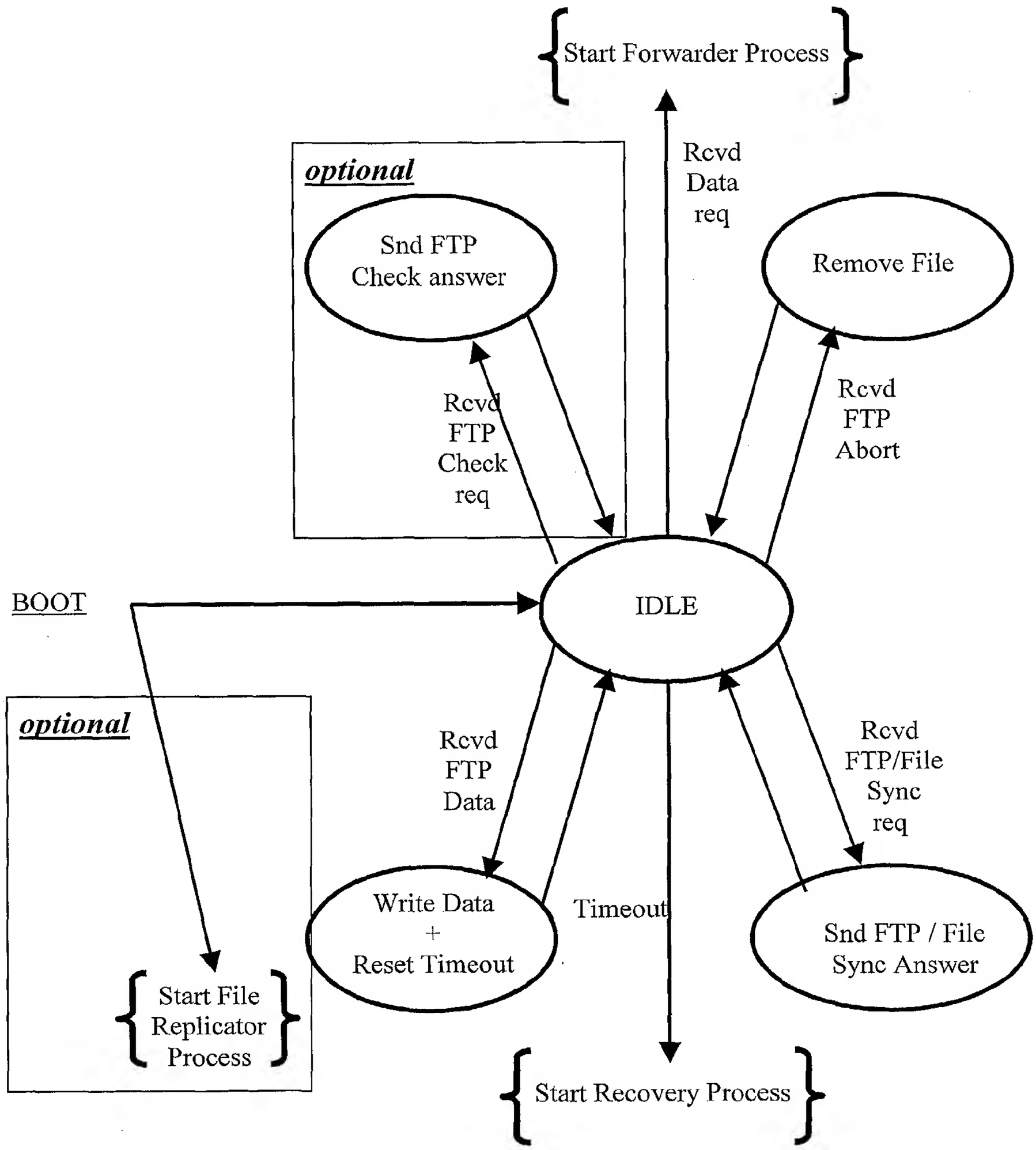


Figure 7: Multicast/Broadcast Master Process Protocol Finite State Machine

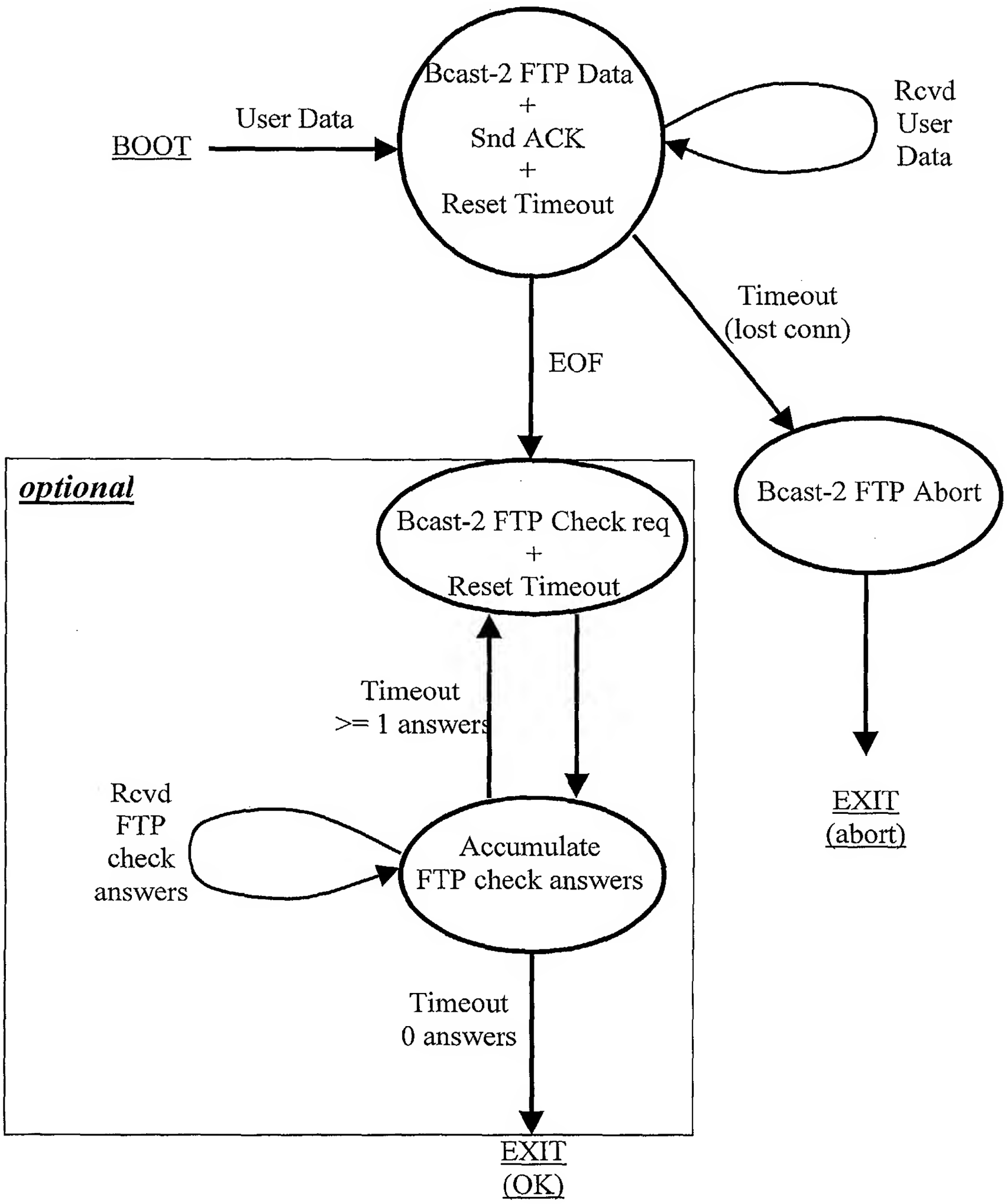


Figure 8: Forwarder Slave Process Protocol Finite State Machine

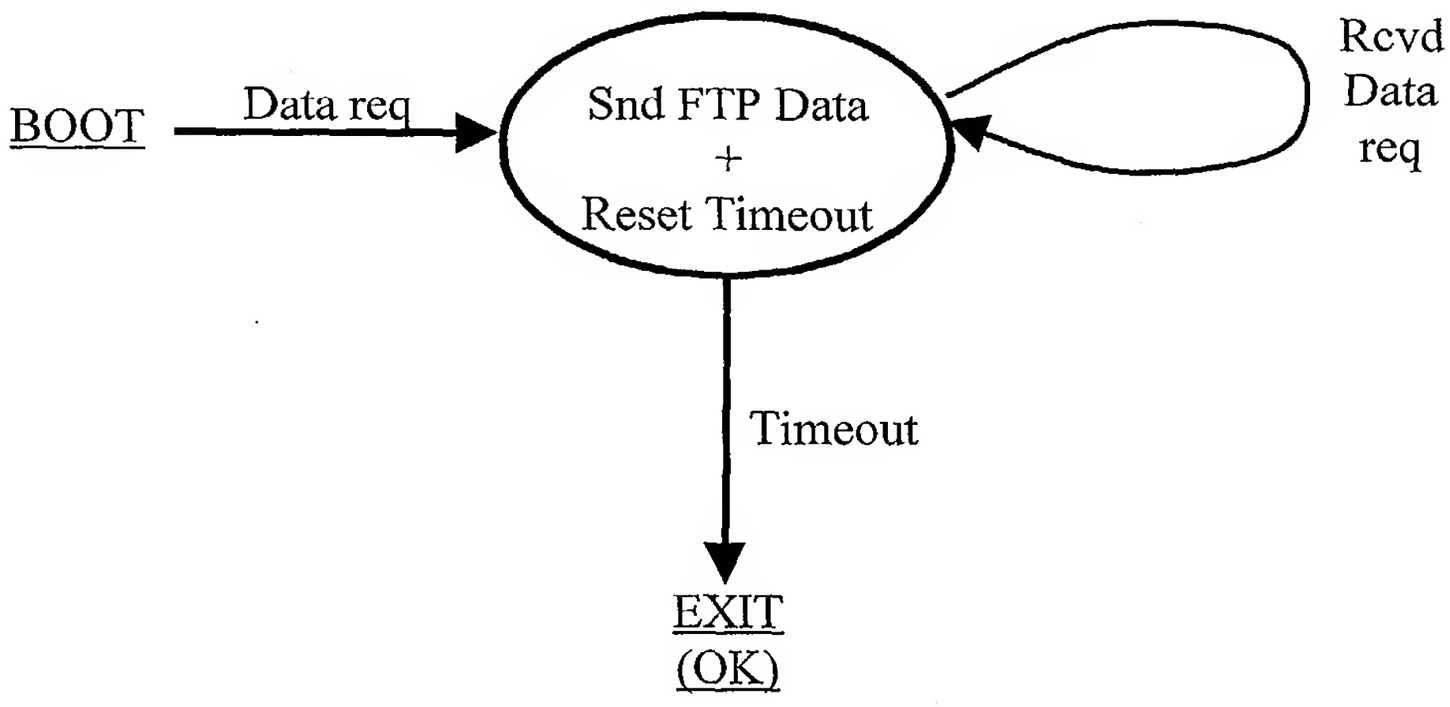


Figure 9: Transfer Error Recovery Slave Process Protocol Finite State Machine

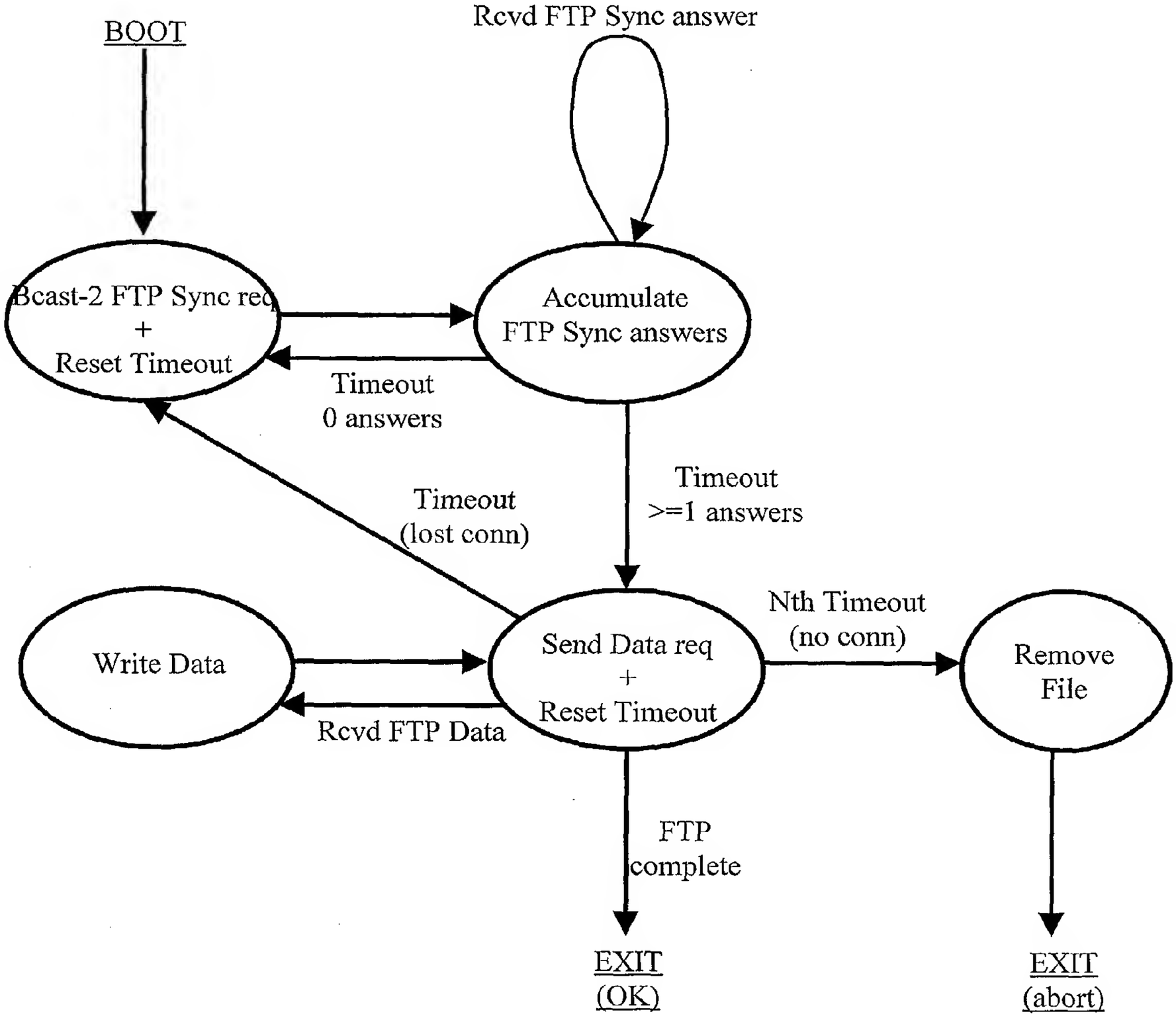


Figure 10: File Replication Slave Process Protocol Finite State Machine

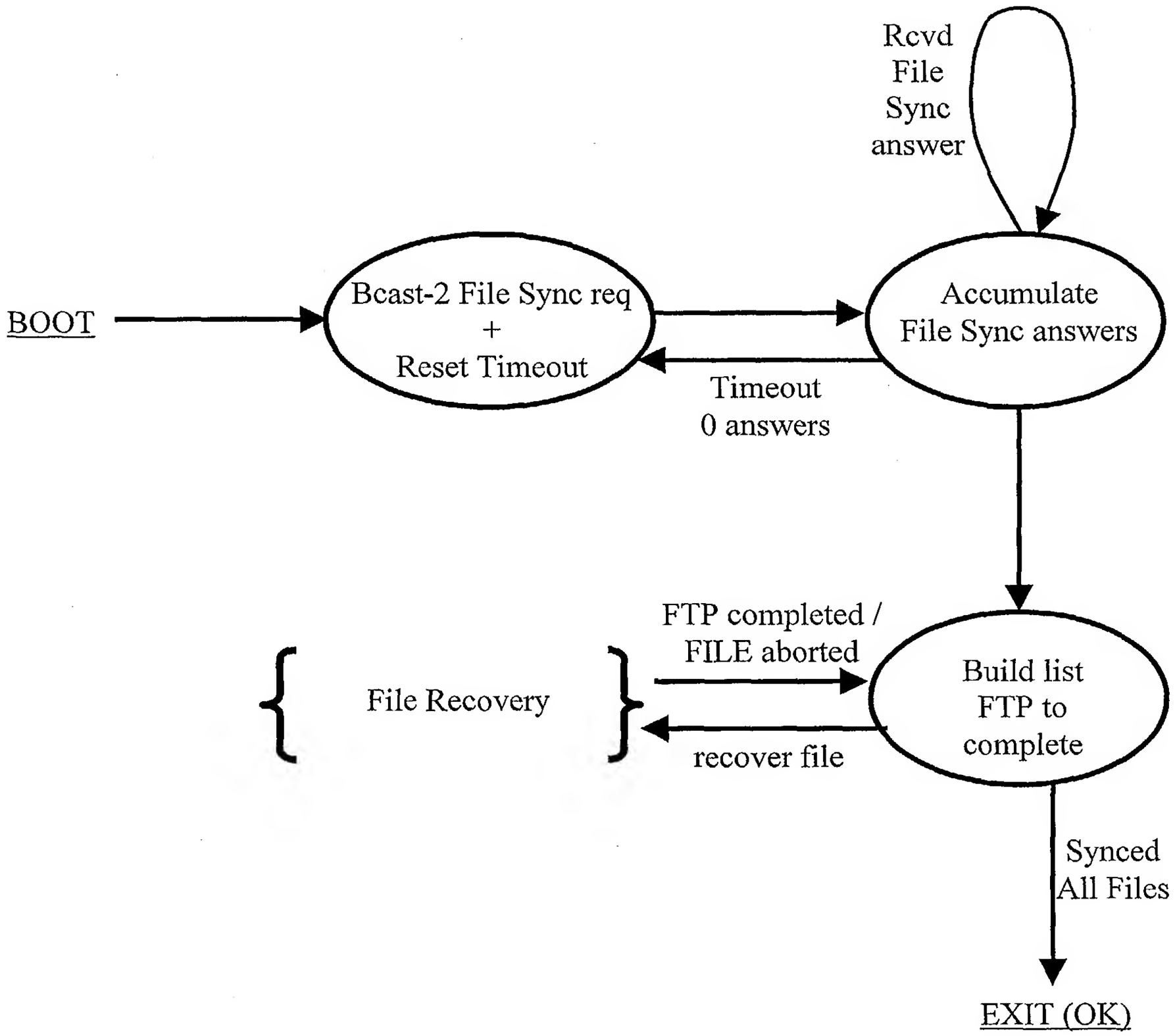
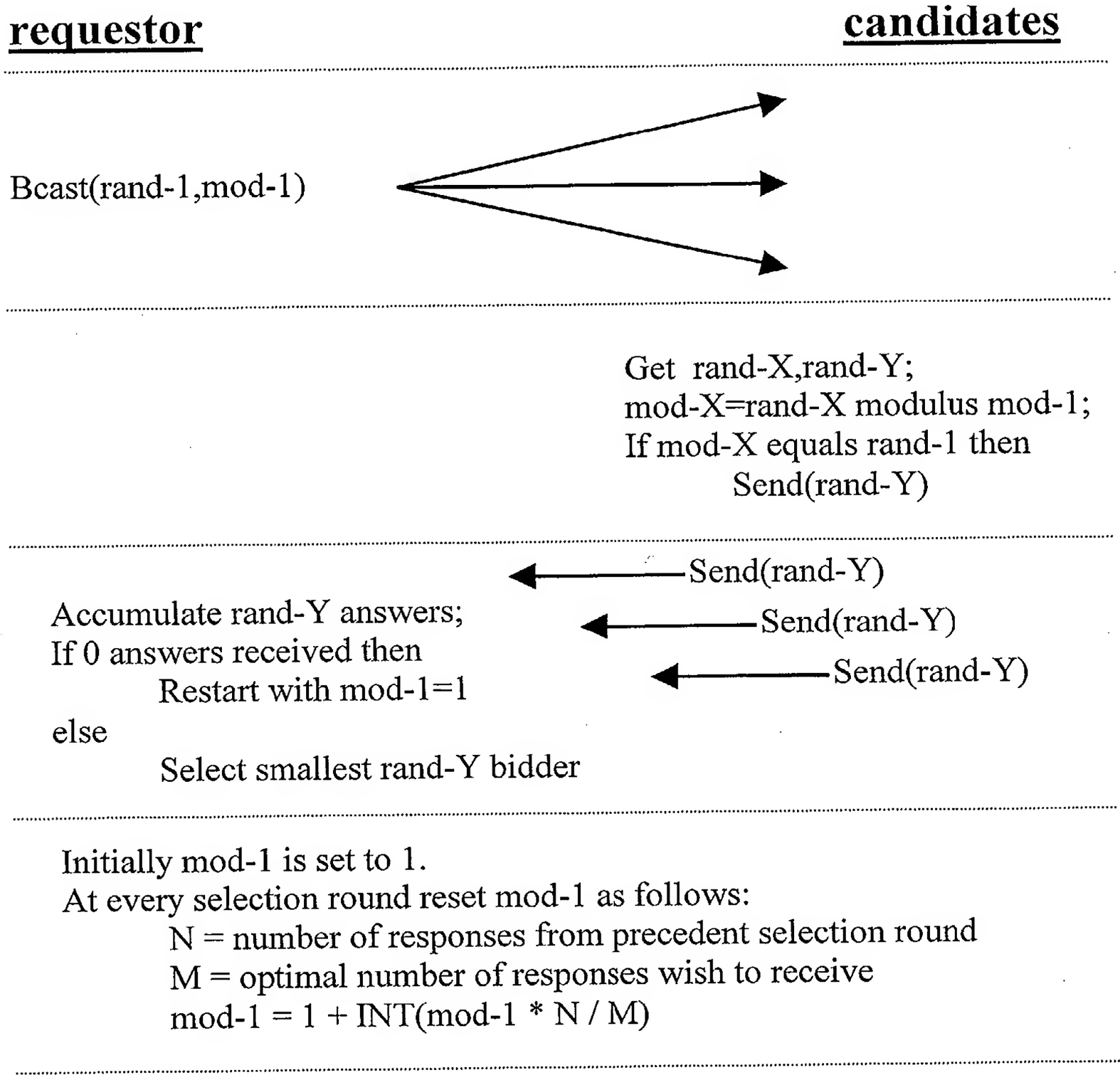


Figure 11: Distributed Selection Mechanism



European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 03 25 3107

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.7)
X	KASERA S K ET AL: "A comparison of server-based and receiver-based local recovery approaches for scalable reliable multicast" INFOCOM '98. SEVENTEENTH ANNUAL JOINT CONFERENCE OF THE IEEE COMPUTER AND COMMUNICATIONS SOCIETIES. PROCEEDINGS. IEEE SAN FRANCISCO, CA, USA 29 MARCH-2 APRIL 1998, NEW YORK, NY, USA,IEEE, US, 29 March 1998 (1998-03-29), pages 988-995, XP010270376 ISBN: 0-7803-4383-2 * page 988, column 1, line 24 - line 27 * * page 988, column 1, line 36 - line 40 * * page 988, column 2, line 12 - line 18 * * page 989, column 2, line 30 - line 33 * * page 990, column 1, line 1 - line 18 * ---	1-9	H04L12/18 H04L29/06
X	BAUER D ET AL: "An error-control scheme for a multicast protocol based on round-trip time calculations" LOCAL COMPUTER NETWORKS, 1996., PROCEEDINGS 21ST IEEE CONFERENCE ON MINNEAPOLIS, MN, USA 13-16 OCT. 1996, LOS ALAMITOS, CA, USA,IEEE COMPUT. SOC, US, 13 October 1996 (1996-10-13), pages 212-221, XP010200690 ISBN: 0-8186-7617-5 * page 213, column 2, last paragraph * * page 214, column 2, line 35 - line 41 * -----	1,2,7	

TECHNICAL FIELDS SEARCHED (Int.Cl.7)

H04L

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	24 September 2003	van der Meulen, E-J

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)